Usability and open-source software development

David M. Nichols, Kirsten Thomson and Stuart A. Yeates
[daven, kthomson, s.yeates]@cs.waikato.ac.nz
Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Abstract

Open-source is becoming an increasingly popular software development method. This paper reports a usability study of the open-source Greenstone Digital Library collection-building software. The problems highlighted by the study are analysed to identify their likely source within the social context of Greenstone's development environment. We discuss how characteristics of open-source software development influence the usability of resulting software products.

Introduction

Open-source is becoming an increasingly popular software development method, producing successful software such as the Linux operating system and the Apache web server. Open-source software is popular with technically sophisticated users, who are often also the software developers, and has not yet made a significant impact on the desktop of most users. This paper starts to address this issue through a usability study of the open-source Greenstone Digital Library Software. The inter-disciplinary nature of Greenstone's target users provide an interesting (and relatively rare) test case for the generality of open-source development methods.

The paper begins with a brief overview of the Greenstone software, and open-source projects in general. We then describe the results of a usability study of collection-building using Greenstone. The problems highlighted by the study are analysed to identify their likely sources within Greenstone's development environment. We then discuss how characteristics of open-source software development influence the usability of the resulting software.

Greenstone

Greenstone is an open-source software system for building, maintaining and serving digital library collections [10]. It runs on a wide variety of platforms and provides full-text mirroring, indexing, searching, browsing and metadata extraction. With features that have evolved out of honours, masters and PhD research projects, most of the development is done within the New Zealand Digital Library (NZDL) research group at the University of Waikato.

No individual decides which features are added to the system, instead a technical consensus among the key developers guides overall direction and individual developers work on features and functionality that are useful to them. A single full-time programmer works on bug-fixing and small external contracts. Valuable use-cases are derived from researchers own research and digital library use presented at conferences and in journals as well as end-user feedback. Although most developers are on-site, external developers (including users) also contribute to the software. A lack of a controlling manager is typical of open source development environments, with developers usually only having to show technical ability and willingness to work on the project before they are granted access to the source code.

Greenstone has two classes of users: those who build collections and those who access collections as end-users. Librarians are an important group of potential collection-builders but Greenstone is aimed at anyone who wants to create a structured repository of documents. Thus, Greenstone's target users are a diverse multi-disciplinary group.

Open-Source Projects

"Open source is a term … to describe the tradition of open standards, shared source code, and collaborative development" [6] behind software such as the Linux operating system. Projects place their source code in the public domain; this allows third parties to contribute code and facilitates the creation of a self-organizing networked community of developers. Raymond [7] describes "treating your users as co-developers" and "given enough eyeballs all bugs are shallow" as key parts of open source development. While this has been shown to work, it changes the role of the user to that of a user-developer.

There have been many successful open-source projects [6,2], e.g. Linux, Apache, Perl, Sendmail, Bind, Tcl/tk and Python. Although users of the Internet indirectly rely on several of these programs, most computer users never *directly* use a piece of open-source software. The success of open-source has been

almost totally confined to a small clique of technically adept users although some see the user base continuously expanding [7]. This may help explain why, to our knowledge, there is no literature on the relationship of user-centered design methods to open-source software development. Indeed, "academic inquiry into OSS [open-source software] is sorely needed" [2]; in this paper we contribute to that inquiry by examining how open-source software development influences usability.

A Usability Study of Collection Building with Greenstone

Greenstone has several typical open-source features: many developers, a fast release-schedule, a non-commercial nature and a lack of explicit project management. However, it is relatively unusual in being a mature open-source project (started in 1995) aimed at non-technical users. These characteristics make it an interesting case to examine in terms of the usability of software developed via an open-source process.

Previous testing of the Greenstone software has examined how end-users search existing collections [8,1]. In contrast, the study reported here examined the two methods Greenstone provides for building collections: the standard command-line interaction and a recently-released alternative, a browser-based tool known as 'The Collector' [9].

Method

The participants, 23 fourth and fifth year Computer Science students, were asked to build collections using the two alternative methods: the command-line and the Collector. Most of the participants had some familiarity with the NZDL as end-users (using collections for their information searching) but the majority were unfamiliar with either of the collection-building processes. The participants had access to the three Greenstone manuals (an *Installer's Guide*, a *User's Guide* and a *Developer's Guide*), in both paper and electronic formats. To reflect the likely environment of a potential remote user we gave no additional explanation of the documentation and chose to use the Windows 98 operating system.

Initially, two sets of URLs were used as the source material for the collections. During the study the presentation of these shortcuts was altered and one data set was changed to Word documents. Some changes arose from observations that some participants confused shortcuts to URLs with saved HTML files. The two collection building methods and data sets were randomised to mitigate any learning effects. The participants were also asked to fill in some short questionnaires: pre-session, post-task and a post-session comparison. Each session took place in the University of Waikato Usability Laboratory and was recorded on video.

Results Summary

Although some video remains to be analysed we can safely say that the participants preferred the Collector and would use it again in preference to the command-line. Questionnaire responses about the command-line process and its documentation were strongly negative – however, task changes during the study complicate comparisons. There were several problems with the consistency of the software and the documentation, particularly when using the command-line. Examples of specific usability issues are described in the next section where they are linked to the software development environment of Greenstone.

An additional 'result' of the study was that several Greenstone developers (of both software and documentation) were able to observe, at first-hand, the participants working with their system. As noted by Nielsen [5], this proved to be a very effective method of communicating the usability issues involved.

What are the underlying causes of the usability issues?

Many of the software usability issues identified by the study deal with fundamental human-computer interaction concepts such as feedback and consistency. However, although identifying the problems has improved the Greenstone software and documentation it is not our main focus here. We are concerned with the design processes which generated, or failed to correct, these problems. Analysis of the video data and discussions with Greenstone project members suggest that the usability issues can be placed into four groups.

Developer Knowledge

Some issues we identified could be traced to explicit pieces of knowledge known by the developers but not by our users. The software and documentation did not reflect this knowledge (the users just had to know) and this led users into errors. One example we observed, using the command-line, concerned the relationship of environment variables to command line windows (they require re-setting for each window).

One user, while trying to build a collection, completed the setup phase but then incorrectly specified the command to start building a collection. He then consulted documentation to understand the problem, closed his command-line window, opened a new window and then gave the correct command. The command failed because ('behind the scenes') the environmental variables were not set correctly. The user then re-issued the setup command. However, he inferred that the failure of the correct command was due to syntax (rather than the system state) and created another incorrect command variant.

The surface explanation for this episode is a lack of feedback from the setup command. As with many usability issues, a piece of 'common knowledge' amongst the developers was not communicated to the users. Specifically for an open-source project, many people had seen this text and behaviour and no-one had suggested that it would cause problems. This is the standard sequence of commands and has been present since the first version of Greenstone on Windows. If all the 'eyeballs' have this piece of 'common knowledge' then it may well not occur to them that anything is amiss. This was not a functional bug to be patched, but an issue of differential knowledge that was not 'shallow' to anyone and was only discovered through testing with non-developers.

Developer Bias

Several issues we identified related to developers' differential use of operating systems and environments. One example of a problem that developers rarely encountered concerned the behaviour of commands with incorrect arguments. The default response for the commands was to display a list of the available options. The length of this text is greater than the default size of the command-line window on Windows 98; this meant that the specific error message scrolled off the top of the top of the screen. Worse, the default window has no scroll bar, meaning that the system's response is effectively inaccessible. This led to many confused users struggling to correct their mistake from memory.

This behaviour appears to have three possible causes, firstly, developers make fewer errors so experience error-behaviours less frequently. Secondly, developers predominantly work in customized environments and so don't experience the effects of default settings. Finally, developers rarely run or test on Windows 98, although Greenstone's multi-platform capabilities are a feature of the software. The combination of these effects insulated the developers from the situation we observed with the users.

Interaction Style

The command-line interaction style, of returning to a prompt with no feedback to indicate success, caused participants several problems. Several users commented on, or were confused by, the lack of feedback from many of the command line operations. Although most commands did provide some output they didn't finish with a message such as 'operation x completed successfully'. Some users attempted to repeat commands believing they hadn't worked correctly while others were just uncertain about the process.

The commands were following the Unix convention of 'return to the prompt indicates success'. Although the users were generally familiar with this convention when using Linux they seemed unsure when it appeared in the context of the study. This may be because the commands they were using were Greenstone-specific rather than generic file management commands. That commands should indicate success 'silently' is a convention that the developers are familiar with—that is how all the usual commands work. The users we observed expected unfamiliar commands to provide explicit success feedback.

Documentation

Several users had problems with the Greenstone documentation; both as a reference source when they encountered difficulties and as a guide to step through a sequence of actions. One notable case occurred when a user interpreted a lack of feedback from a command as an error. The following paragraph in the documentation provided recovery instructions from a different type of error. However, the user applied these commands even though he was aware they were intended for a different situation, seemingly on the grounds that something was wrong and these were adjacent recovery instructions from an error. The links between the errors and the fixes are understood by the developers but, in the documentation, only the surface proximity was available to the participants.

The documentation for the Collector contains examples of how to specify the source data for a collection. However, the file path example given is Unix-specific and this forced the participants to adapt

the example for a Windows platform without crucial case-sensitivity and 'slash' direction information (confusingly for the participants, forward slashes were required to specify Windows files). In this case the platform bias of the developers was reflected in the documentation rather than in the software.

A further example concerned a command that required an email as a option, the manual gave an example with the option as `<your_email>`. Several of our users typed the angle brackets; a trivial error that caused significant delays but one that was completely invisible to the developer community. The typography and structure of the documentation caused other problems highlighting that, although Greenstone is well-documented by open-source standards, the development team contains no professional technical writers.

Open-Source and Usability

The problems we observed were typical of usability issues that frustrate novice users. Many of the identified issues had been present in the Greenstone software and documentation for some considerable time. They had been effectively invisible to the developers on the project and had not been reported by the users. Although Greenstone is not a completely typical open-source project we wondered whether there are structural reasons inside open-source projects that could cause these types of usability issues to persist.

The central mechanism for achieving software quality in open-source projects is extensive beta-testing [7]. This 'bazaar-style' of development successfully encourages extensive functional testing of error-prone software to produce robust and reliable software such as the Apache web server [4]. However, elements of usability may not be equally well-supported by open-source development—particularly when applied to software aimed at less technically-sophisticated users.

User-Involvement: Developers are not Typical Users

"For this open source approach to work, large numbers of users have to be both interested in and capable of debugging source code" [3]. This approach works well when the users can be expected to be relatively technically sophisticated (e.g. Linux, Apache, Perl etc). For projects with less technical target users (e.g. Greenstone) then it is unlikely that most users can debug source code; indeed they will probably need support in simply reporting bugs.

That the developers are not users (and so are not representative of users) is one of the core elements of usability engineering [5]—yet the substantial overlap between developers and users in open source development collapses this distinction. In our study we found numerous examples of behaviour and documentation that the developers were happy with—but which caused significant problems for our participants (who had more technical experience than many of Greenstone's target users). We suspect that some users would have encountered these problems, given up and tried other software. In contrast, a typical open-source user-developer would have at least reported the problem and maybe suggested a solution. We hypothesize, for software such as Greenstone, that most users are unlikely to contribute to the development process.

The Culture of Open-Source Development

"The utility function Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers" [7]. The social context of open-source software development does not include the typical user and so it may be unreasonable to expect their interests to be represented in the programs. As work on open-source projects is voluntary then developers work on the topics that interest them and this may well not include features for novice users. We suspect that 'hard' algorithmic problems have a greater value in the 'reputation market' than issues of usability. In addition, these functional problems are easier to specify, evaluate and modularize; all attributes which simplify decentralized problem solving.

Ironically, the *open*-source developer culture of hackers can appear (from the outside) to be a closed idiosyncratic community. Increasingly, software design is an inter-disciplinary effort and there is little evidence so far of input from outside the developer community into open-source software. The successful integration of user-centered methods such as field studies and ethnographic observations could be a significant step towards generalizing the open-source development model.

Resources

The Greenstone project generates its own documentation and does not employ professional technical writers. Similarly, open-source projects do not tend to have the resources to run detailed usability studies—the study reported here was facilitated by the co-location of the NZDL and newly constructed usability facilities. It is noticeable that in end-user testing of several digital library systems [1] the open-source Greenstone was compared against well-funded commercial systems such as Ingenta (www.ingenta.com) which do not attempt to provide collection-building facilities. The availability of dedicated usability facilities enabled Greenstone developers to observe users using the system in controlled circumstances, an opportunity most open-source developers rarely get due to the de-centralized nature of their projects.

On the other hand, a de-centralized project could be expected to have more diversity in operating systems and machine configurations. This diversity would tend to reduce the effects of some of the platform-specific development issues that we observed with Greenstone.

Conclusion

We have described the results of usability testing of the Greenstone collection-building software. When we examined the types of problems we found they are typical of 'classic' issues of usability (the differences between developers and users). Although Greenstone has benefited from contributions from many people these issues had never been addressed, for a variety of reasons: resources, motivation, access to non-technical users and the development model adopted.

Our experience with Greenstone suggests that open-source development methods may need to adapt if they are to produce software for the desktop of the typical user. A community of developers will not necessarily pay sufficient attention to issues of usability that they themselves do not experience. An interesting question is whether a large open-source project could address usability issues without the well-known benefits of studying real users?

Acknowledgements

We would like to thank the students who participated in the study and members of the New Zealand Digital Library project for their cooperation. Michael Twidale provided useful comments on a draft of the paper.

References

1. Blandford, A., Stelmaszewska, H. & Bryan-Kinns, N. (2001 to appear) Use of multiple digital libraries: a case study, *Proceedings of the First Joint Conference on Digital Libraries*, Roanoke, VA.
2. Feller, J. & Fitzgerald, B. (2000) A framework analysis of the open source software development paradigm, *Proceedings of the 21st International Conference on Information Systems*, Brisbane, Australia. ACM Press. 58-69.
3. McConnell, S. (1999) Open-source methodology: ready for prime time?, *IEEE Software*, 16(4), 6-8.
4. Mockus, A., Fielding, R.T. & Herbsleb, J. (2000) A case study of open source software development: the Apache server, *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland. ACM Press. 263-272.
5. Nielsen, J. (1993) *Usability Engineering*. Academic Press, Boston, MA.
6. O'Reilly T. (1999) Lessons from open-source software development, *Communications of the ACM*, 42(4), 32-37.
7. Raymond, E.S. (1998) The cathedral and the bazaar, *First Monday*, 3(3), [www.firstmonday.org].
8. Theng, Y.L., Duncker, E., Mohd-Nasir, N., Buchanan, G. & Thimbleby, H. (1999) Design guidelines and user-centred digital libraries, *Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries (ECDL'99)*. Springer-Verlag, 167 – 183.
9. Witten, I.H., Bainbridge, D. & Boddie, S.J. (2001 to appear) Power to the People: end-user building of digital library collections, *Proceedings of the First Joint Conference on Digital Libraries*, Roanoke, VA.
10. Witten, I.H., Boddie, S.J., Bainbridge, D. & McNab, R.J. (2000) Greenstone: a comprehensive open-source digital library software system, *Proceedings of the Fifth ACM Conference on Digital Libraries*, San Antonio, TX. 113-121.